Adaptation dynamique de codes parallèles

Jérémy Buisson — Françoise André — Jean-Louis Pazat

IRISA/INSA de Rennes/Université de Rennes 1
Campus universitaire de Beaulieu
Avenue du Général Leclerc
35042 RENNES Cedex - France
{jeremy.buisson,francoise.andre,jean-louis.pazat}@irisa.fr

RÉSUMÉ. *Parce que les architectures de type grille de calcul sont très dynamiques, utiliser efficacement leurs ressources est difficile. Les applications doivent pouvoir réagir dynamiquement aux changements de l'environnement d'exécution sous-jacent. Pour aider les développeurs dans la construction d'applications pour grilles de calcul, nous cherchons à définir un modèle d'adaptation de composants parallèles. Cet article se concentre sur les mécanismes d'adaptation fournis aux composants. Nous décrivons en quoi une plate-forme générique peut aider au développement d'applications efficaces pour grilles de calcul ; des résultats expérimentaux montrent les apports de l'utilisation d'une telle plate-forme.*

ABSTRACT. *Since grid architectures are known to be highly dynamic, using resources efficiently on such architectures is a challenging problem. Software must be able to dynamically react to the changes of the underlying execution environment. In order to help developers to create software for the grid, we are investigating a model for the adaptation of parallel components. This paper focuses on the adaptation mechanisms that are provided as a meta-level for components. We describe how a generic platform can help to develop efficient grid software. First experimental results show the gain that can be expected from the use of such a platform.*

MOTS-CLÉS : *auto-adaptation dynamique, composants, parallélisme*

KEYWORDS: *dynamic self-adaptation, components, parallelism*

1. Introduction

La plupart des projets d'étude des grilles de calcul se sont focalisés, à l'image de OGSA [FOS 02] et Globus [Glo], sur la manière dont les plate-formes peuvent mettre les ressources à la disposition des applications. Peu de projets étudient comment créer des applications efficaces pour ces architectures. Des projets tels que GridCCM [PéR 02] cherchent à transposer les techniques existantes de couplage de codes ; ils se sont généralement concentrés sur l'efficacité des communications dans le cadre de modèles de programmation classiques à base de composants. Il manque cependant une approche globale à cette problématique.

La volatilité et la grande variabilité des ressources en environnements mobiles ont conduit à l'étude de la notion d'adaptabilité. Il s'agit de modifier le comportement des applications en fonction de l'environnement d'exécution ; cela va du paramétrage au remplacement de leur implémentation. Les ressources des grilles de calcul sont caractérisées par leur variabilité et leur imprévisibilité dues à leur partage entre les applications et aux reconfigurations du matériel. Les techniques d'adaptation se développent donc également dans ce cadre. Plusieurs moyens existent pour l'adaptation. Par exemple, il est possible de sélectionner une mise en œuvre parmi plusieurs : ce choix peut être guidé par des règles données explicitement dans une politique d'adaptation comme dans ACEEL [CHE 03] ; ou bien en comparant les modèles de performances des différentes mises en œuvre disponibles tel que dans ICENI [FUR 02]. Une autre approche consiste à se baser sur des mécanismes d'intercession pour modifier directement le comportement tel que ceci est permis par exemple dans PCL [ENS 02].

La problématique que nous proposons d'étudier est la combinaison des techniques d'adaptation dynamique avec les spécificités du parallélisme à grande échelle pour grilles de calcul. Comme première étape de notre étude, nous nous concentrons sur les mécanismes d'adaptation au sein des composants. Nous nous intéressons tout particulièrement à la synchronisation des différents codes – fonctionnels et d'adaptation – et à la coordination des différents processus des composants parallèles.

2. Modèle d'applications pour grilles de calcul

Dans notre modèle, une application est un assemblage de composants ; chacun de ses composants est déployé sur une machine parallèle qui résulte de l'agrégation des ressources allouées par le mécanisme d'allocation et d'ordonnancement. Les ressources allouées peuvent donc changer dynamiquement si on veut rendre possible la réallocation des ressources pendant l'exécution. Le mécanisme d'adaptation que nous proposons permet à chaque composant de tenir compte à chaque instant des ressources qui lui sont effectivement allouées. Les composants qui forment l'application sont parallèles et auto-adaptables : ils encapsulent plusieurs processus concurrents qui collaborent ; il sont capables de modifier leur comportement compte tenu de leur perception de l'environnement d'exécution.

D'une manière générale, les mécanismes d'adaptation dynamique fonctionnent tous de manières similaires : quand l'environnement d'exécution change, un moniteur génère un événement qui est envoyé au décideur. Celui-ci se base alors sur la politique d'adaptation pour déterminer s'il doit s'adapter et quelle réaction il doit exécuter pour modifier le comportement. Ceci est illustré par la figure 1. Chaque comportement est une collection de composants internes séquentiels qui collaborent à l'accomplissement du service que le composant doit rendre ; les composants internes communiquent entre eux en utilisant un paradigme tel que le passage de messages (par PVM ou MPI par exemple) ou la mémoire partagée distribuée.

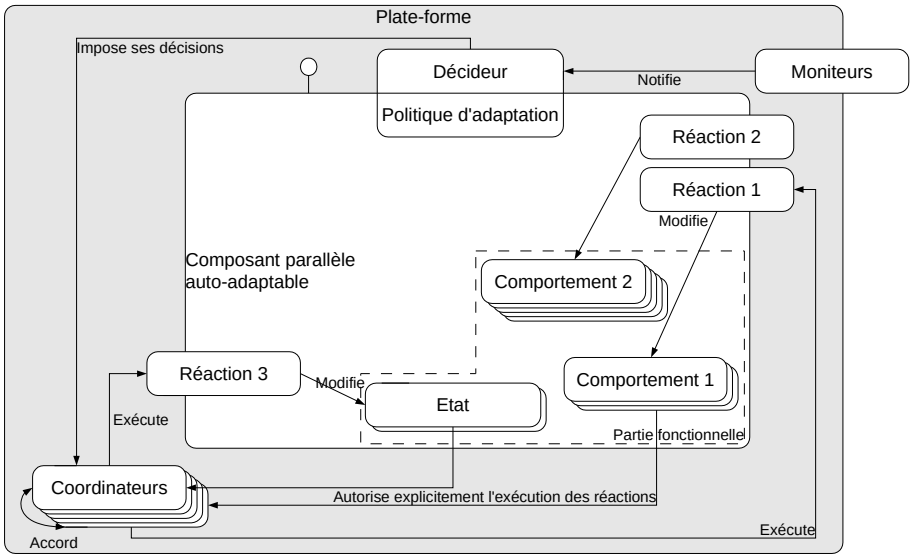

Figure 1. *Un composant parallèle auto-adaptable déployé dans sa plate-forme*

2.1. *Réaction*

Une réaction est un « moyen » pour le composant de s'adapter : c'est une portion de programme mise à la disposition du mécanisme d'adaptation pour agir sur le comportement du composant dans le but de modifier sa manière d'utiliser les ressources. Remplacer l'implémentation par une autre, paramétrer le composant, modifier le degré de parallélisme sont des exemples de réactions. Ces réactions peuvent être mises en œuvre par exemple par programmation réflexive, ou bien – pour remplacer l'implémentation – en se basant sur le patron de conception *Strategy* [GAM 98] comme dans ACEEL.

2.2. *Point d'adaptation*

Un point d'adaptation exprime « quand » un composant peut s'adapter : c'est une position dans le code du composant à partir de laquelle il est possible d'exécuter une réaction, c'est-à-dire un état qui vérifie les préconditions imposées par la réaction et que la réaction transforme en un état « équivalent » d'un autre comportement.

Dans le cas parallèle, un point d'adaptation global est une combinaison de points d'adaptation locaux à chacun des composants internes. Toutes les combinaisons ne représentent pas des points d'adaptation globaux : seules certaines correspondent à un état atteignable. Il faut remarquer que le fait que deux ensembles de points d'adaptation de deux composants internes vérifient localement les préconditions d'une réaction n'implique pas que toutes leurs combinaisons vérifient la conjonction des préconditions de cette réaction. Il faut donc coordonner l'ensemble des composants internes pour garantir que l'adaptation ne se produit que sur des points d'adaptation globaux.

Pour mettre en œuvre les points d'adaptation, on peut imposer au développeur de placer des directives dans son code pour indiquer leur position. Une autre approche serait de se baser sur les techniques de programmation par aspects [KIC 97] : l'adaptabilité étant un aspect dont les points de jonction sont les points d'adaptation, le tissage de cet aspect placerait les directives de manière automatique.

2.3. *Rôle de la plate-forme*

Le développeur fournit des mises en œuvre de son composant (les comportements), des réactions et une politique d'adaptation ; les comportements définissent des points d'adaptation. La plate-forme – composée du décideur et des coordinateurs – fait le lien entre ces différents éléments : elle ordonnance l'exécution des réactions.

La politique d'adaptation indique « quand » et « comment » le composant *doit* s'adapter ; les points d'adaptation spécifient « quand » dans le fil de l'exécution du composant et « comment » le composant *peut* s'adapter. Le rôle de la plate-forme est donc de trouver un compromis entre les indications de la politique d'adaptation et les contraintes traduites par les points d'adaptation.

3. Expérimentation

Pour valider le modèle qui a été proposé, nous avons développé une plate-forme de test qui met en œuvre un sous-ensemble des fonctionnalités décrites.

Voici un exemple classique d'adaptation qui est souvent réalisé de manière ad hoc. L'application est un code itératif générique sur des vecteurs distribués par blocs qui utilise MPI comme moyen de communication interne. La politique d'adaptation est d'utiliser autant de nœuds de calcul que le moniteur en indique de disponibles : lorsqu'un nœud est ajouté au système, l'application démarre un nouveau processus ;

lorsqu'un nœud doit être retiré du système, l'application termine les processus qui s'exécutent dessus. L'implémentation de MPI que nous utilisons étant incapable de démarrer ou terminer dynamiquement des processus, cette fonctionnalité est simulée, de même que la surveillance du nombre de nœuds du système. Les points d'adaptation sont placés entre chaque itération.

Nous avons comparé la version originale de cette application à sa version adaptable dans un environnement dans lequel le nombre de nœuds disponibles passe de 4 à 6. On mesure les temps d'exécution à la fin de chaque itération ; ils sont reportés sur la figure 2. La réaction, qui consiste à démarrer deux processus et à redistribuer les vecteurs, apparaît par une cassure de la courbe entre les itérations 12 et 13. Pour compenser l'exécution de cette réaction, plusieurs itérations sont nécessaires. Leur nombre dépend de la réaction exécutée et de la nature du composant. Cependant, le gain de parallélisme obtenu à la fin de l'exécution permet de réduire significativement le temps total d'exécution. Il est intéressant de voir que ce type d'adaptation est utile et entre très bien dans le cadre de ce que nous avons décrit.

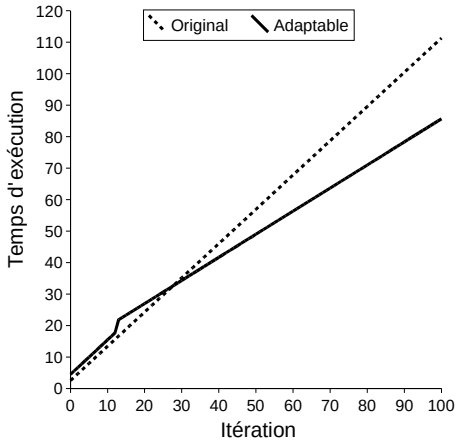

Figure 2. *Mesure du temps d'exécution à la fi n de chaque itération*

4. Conclusion

Nous avons montré que combiner adaptation dynamique et composants parallèles est une idée prometteuse pour faciliter la construction d'applications efficaces pour des architectures de type grille de calcul. Un prototype basé sur le framework ACEEL [CHE 03] a permis d'expérimenter nos idées sur le sujet.

Dans la suite de nos travaux, nous allons étudier et formaliser la notion de point d'adaptation. L'objectif est d'être capable de vérifier leur placement. A plus long terme, on aimerait pouvoir placer automatiquement des points d'adaptation pour une

classe d'applications donnée. Il sera en particulier intéressant d'étudier les relations entre les points d'adaptation et les points de reprise définis dans les mécanismes de tolérance aux fautes.

Dans la suite de notre étude, nous intégrerons des applications en vraie grandeur pour lesquelles l'adaptabilité est vitale. Nous avons d'ores et déjà exploré quelques pistes telles que le logiciel AMAP [AMA] de simulation de la croissance des plantes. Il faudra pour cela étudier l'adaptation coordonnée de composants qui coopèrent.

Notre objectif est d'obtenir à terme une plate-forme générique de composants parallèles adaptables pour grilles de calcul. Cette plate-forme combinera les outils d'aide au développement de ces composants ainsi que le support nécessaire à leur exécution. Une telle plate-forme devrait grandement faciliter le développement d'applications efficaces pour grilles de calcul.

5. Bibliographie

[AMA] « Atelier de Modélisation de l'Architecture des Plantes », http ://wwww.cirad.fr/presentation/programmes/amap/logiciels/amap.shtml.

[CHE 03] CHEFROUR D., ANDRÉ F., « Développement d'applications en environnements mobiles à l'aide du modèle de composant adaptatif ACEEL », *Langages et Modèles à Objets LMO'03. Actes publiés dans la Revue STI, série L'objet, volume 9*, Vannes, France, février 2003.

[ENS 02] ENSINK B., ADVE V., « Language Support for Coordinating Adaptation in Distributed Systems », rapport n° UIUCDCS-R-2002-2309, décembre 2002, Computer Science Department, University of Illinois at Urbana-Champaign.

[FOS 02] FOSTER I., KESSELMAN C., NICK J., TUECKE S., « The Physiology of the Grid : An Open Grid Services Architecture for Distributed Systems Integration », *Global Grid Forum*, juin 2002.

[FUR 02] FURMENTO N., MAYER A., MCGOUGH S., NEWHOUSE S., FIELD T., DARLINGTON J., « ICENI : Optimisation of component applications within a Grid environment », *Parallel Computing*, vol. 28, n° 12, 2002, p. 1753–1772.

[GAM 98] GAMMA E., HELM R., JOHNSON R., VLISSIDES J., *Design patterns : elements of reusage object-oriented software*, Addison Wesley, 1998.

[Glo] « Globus Toolkit », http ://www.globus.org.

[KIC 97] KICZALES G., LAMPING J., MENDHEKAR A., MAEDA C., LOPES C., LOINGTIER J.-M., IRWIN J., « Aspect-Oriented Programming », AKŞIT M., MATSUOKA S., Eds., *Proceedins of European Conferebce on Object-Oriented Programming*, vol. 1241, Berlin, Heidelberg and New-York, 1997, Springer-Verlag, p. 220–242.

[PéR 02] PÉREZ C., PRIOL T., RIBES A., « A parallel CORBA component model », Rapport de recherche n° 4552, septembre 2002, INRIA.